(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 August 2001 (30.08.2001)

PCT

(10) International Publication Number
WO 01/63434 A1

(51) International Patent Classification[7]: G06F 15/16

(21) International Application Number: PCT/US01/06004

(22) International Filing Date: 23 February 2001 (23.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/184,560 24 February 2000 (24.02.2000) US

(71) Applicant: **BOPS, INCORPORATED** [US/US]; 6340 Quadrangle Drive, Suite 210, Chapel Hill, NC 27514 (US).

(72) Inventor: **BARRY, Edwin, Frank**; 1208 Larkhall Court, Cary, NC 27511 (US).

(74) Agent: **PRIEST, Peter, H.**; Priest & Goldstein, PLLC, 529 Dogwood Drive, Chapel Hill, NC 27516 (US).

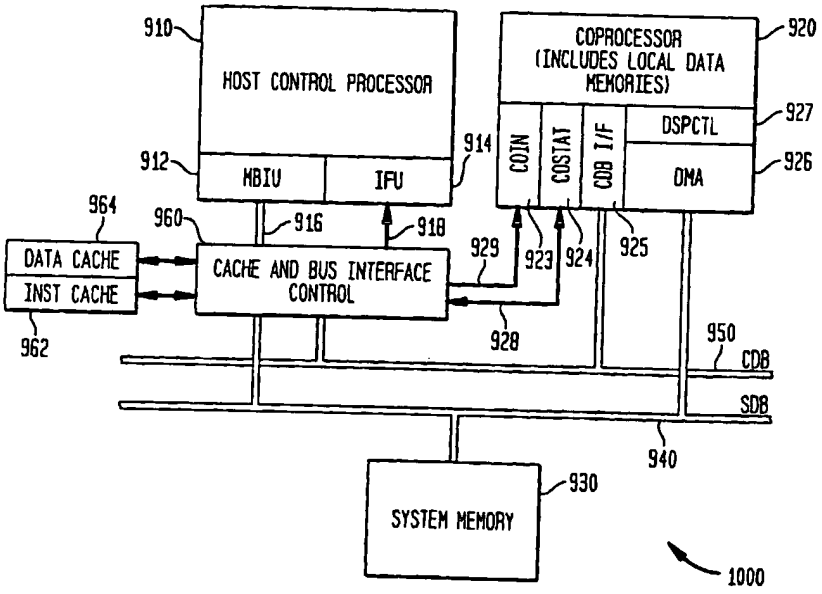
(81) Designated States *(national)*: CA, IL, JP, KR.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHODS AND APPARATUS FOR DUAL-USE COPROCESSING/DEBUG INTERFACE



(57) Abstract: A coprocessor (920) interface (928, 929, 940, 950) is described which provides a flexible degree of coupling with a host processor (910). Specific methods are defined for architectures to make use of the interface for supporting client-server co-processors (CSCOPs). A dynamic debug interface is used to provide a coprocessor interface which supports tightly coupled, loosely coupled and firmly coupled operation.

WO 01/63434 A1

METHODS AND APPARATUS FOR DUAL-USE COPROCESSING/DEBUG INTERFACE

The present invention claims the benefit of U.S. Provisional Application Serial No. 60/184,650 entitled "Methods and Apparatus for Flexible Strength Coprocessing Interface" filed February 24, 2000 which is incorporated by reference herein in its entirety.

Field of the Invention

The present invention relates generally to improvements in coprocessing interfaces and more particularly to advantageous techniques for providing a flexible degree of coupling between a host processor and a digital signal processor.

Background of the Invention

A multiprocessor system consists for two or more processors that communicate to accomplish some task. The processors in the multiprocessor system may or may not be the same. The communications delay between the processors can be considered as representing the coupling strength between the processors. The communications delay represents the time required for a host or control processor to dispatch an operation or command to a coprocessor and for that coprocessor to initiate a response to it. A loosely coupled multiprocessor system usually has a relatively long communications delay as compared to a tightly coupled multiprocessor system that typically has a relatively short communications delay.

There is a class of processors that is described as coprocessors that may not be able to fetch their own instructions but use a "host" processor to supply application specific instructions to the coprocessor. The purpose of the coprocessor is to provide better performance for specialized tasks than could be obtained by the "host" processor acting alone. There is also a class of processors with specialized capabilities, such as digital signal processors (DSPs), that may act as a coprocessor to a control processor. For a number of complex applications, an efficient control processor and an efficient DSP are coupled together to provide an efficient overall solution. It will be recognized that an efficient coupling mechanism is necessary to make a control processor and a DSP system an effective system.

Summary of the Invention

The ManArray scalable family of core processors provides a dual use mechanism for debug support and for a general coprocessor interface. The features of the debug interface can be envisioned to be equally applicable to a coprocessor interface. For an exemplary ManArray processor, the following features of debug support are provided: processor reset

control, instruction fetch control, external or internal (monitor) based debug control, read/write registers, read/write instruction/data memory, read/write VLIW memory (VIM), single-step operation, instruction address breakpoint events, and data address breakpoint events.

There are two standard approaches to achieving a high level of observability and controllability of hardware for debug purposes. One involves the use of scan chains and clock-stepping along with a suitable hardware interface, such as defined by the Joint Task Action Group (JTAG) standard, to a debug control module which supports basic debug commands. This approach allows access on a cycle-by-cycle basis to any resources included in the scan chains, usually registers and memory. It relies on the process technology to support the scan chain insertion and may change with each implementation. The second approach uses a resident debug monitor program which may be linked with an application or resides in on-chip ROM. Debug interrupts may be triggered by internal or external events and the monitor program then interacts with an external debugger to provide access to internal resources using the instruction set of the processor.

The approach proposed here is similar to the debug monitor approach, but allows for debug without a debug monitor program being loaded with, or prior to, the application code. This approach provides a dynamic debug monitor, in which the debug monitor code is dynamically loaded into the processor and executed on any debug event which stops the processor, such as a breakpoint or "stop" command. The debug monitor code is unloaded when processing resumes. This approach includes the static debug monitor as a subset of its operation, but also provides some of the benefits of fully external debug control found in the scan-chain approach.

This dynamic debug interface may be used to provide a coprocessor interface which supports tightly coupled, loosely coupled and firmly coupled operation. One exemplary system for implementing the present invention contains at least two processors. One processor is a ManArray processor operating as the system's coprocessor and the other is a control type processor such as an ARM, MIPS, X86, PowerPC or the like. Tightly coupled operation in this system context means that the coprocessor receives all of its instructions from the control processor. Sometimes a tightly coupled coprocessor is called a "slave processor" since it does not have an independent means of fetching its instructions. Specifically, in this tightly coupled system, an instruction which is not part of the host control processor's instruction set, is dispatched to and accepted by the coprocessor and the control processor does not continue processing further instructions of its own until the coprocessor has completed execution of its instruction.

Loosely coupled operation in this system context means that the host processor dispatches a signal or message to the coprocessor, which then, by executing its own instruction sequence, interprets the message or responds to the signal. The coprocessor then may execute a further sequence of instructions, for example, a subroutine, depending on the message value or signal type. Both the interpretation of a message and subsequent instruction execution based on this interpretation by the coprocessor are carried out concurrently with the host processor's own instruction execution. When a coprocessor subroutine is completed, the coprocessor typically signals completion back to the control processor, and waits for the control processor to send another message to initiate another coprocessor subroutine. In loosely coupled processing, the coprocessor uses its own instruction fetch unit and instruction memory to execute programs.

Firmly coupled coprocessing in this system context specifies that the coprocessor can optionally fetch a sequence of instructions from an instruction first-in-first-out (FIFO) buffer that is allocated a portion of the instruction memory address space. The coprocessor has its own instruction address register, or program counter (PC), and may be directed by a host processor to execute a concurrent subroutine by placing a call or branch-type instruction into the coprocessor's instruction FIFO. When this subroutine is complete, the coprocessor branches to the address of the instruction FIFO. If more instructions are present, then they are executed, otherwise the coprocessor stalls and waits for further instructions. Use of the firmly coupled approach allows the same interface to be used for coprocessors which have a PC and those which do not. If no PC is present in the coprocessor, all the coprocessor instructions are fetched from the coprocessor FIFO.

This coprocessor interface can be designed to advantageously support tightly coupled processing through interlocking with the host processors instruction fetch/decode unit, loosely coupled coprocessing in a coprocessor with a PC, and firmly coupled coprocessing which provides the features of both loose and tight coupling in a common mechanism.

These and other aspects and advantages of the present invention will be apparent from the drawings and the Detailed Description which follow.

Brief Description of the Drawings

Fig. 1 illustrates an exemplary ManArray DSP and DMA subsystem appropriate for use with this invention;

Fig. 2 illustrates a representative coprocessor together with a debug test module in accordance with the present invention;

Fig. 3 illustrates a coprocessor and debug port interface for sharing the ManArray instruction decode register between two possible sources of instructions in accordance with

the present invention;

Fig. 4 illustrates a debug instruction register (DBIR) in accordance with the present invention;

Fig. 5 illustrates a debug status register (DBSTAT) in accordance with the present invention;

Fig. 6 illustrates a DSP control register (DSPCTL) in accordance with the present invention;

Fig. 7 illustrates a debug data out register (DBDOUT) in accordance with the present invention;

Fig. 8 illustrates a debug data in register (DBDIN) in accordance with the present invention;

Fig. 9 illustrates a tightly-coupled coprocessing system in accordance with the present invention;

Fig. 10 illustrates a loosely-coupled coprocessing system in accordance with the present invention; and

Fig. 11 illustrates a firmly-coupled coprocessing system in accordance with the present invention.

Detailed Description

Further details of a presently preferred ManArray core, architecture, and instructions for use in conjunction with the present invention are found in U.S. Patent Application Serial No. 08/885,310 filed June 30, 1997, now U.S. Patent No. 6,023,753, U.S. Patent Application Serial No. 08/949,122 filed October 10, 1997, U.S. Patent Application Serial No. 09/169,255 filed October 9, 1998, U.S. Patent Application Serial No. 09/169,256 filed October 9, 1998, U.S. Patent Application Serial No. 09/169,072 filed October 9, 1998, U.S. Patent Application Serial No. 09/187,539 filed November 6, 1998, U.S. Patent Application Serial No. 09/205,558 filed December 4, 1998, U.S. Patent Application Serial No. 09/215,081 filed December 18, 1998, U.S. Patent Application Serial No. 09/228,374 filed January 12, 1999 and entitled "Methods and Apparatus to Dynamically Reconfigure the Instruction Pipeline of an Indirect Very Long Instruction Word Scalable Processor", U.S. Patent Application Serial No. 09/238,446 filed January 28, 1999, U.S. Patent Application Serial No. 09/267,570 filed March 12, 1999, U.S. Patent Application Serial No. 09/337,839 filed June 22, 1999, U.S. Patent Application Serial No. 09/350,191 filed July 9, 1999, U.S. Patent Application Serial No. 09/422,015 filed October 21, 1999 entitled "Methods and Apparatus for Abbreviated Instruction and Configurable Processor Architecture", U.S. Patent Application Serial No. 09/432,705 filed November 2, 1999 entitled "Methods and Apparatus for Improved Motion

Estimation for Video Encoding", U.S. Patent Application Serial No. 09/471,217 filed December 23, 1999 entitled "Methods and Apparatus for Providing Data Transfer Control", U.S. Patent Application Serial No. 09/472,372 filed December 23, 1999 entitled "Methods and Apparatus for Providing Direct Memory Access Control", U.S. Patent Application Serial No. 09/596,103 entitled "Methods and Apparatus for Data Dependent Address Operations and Efficient Variable Length Code Decoding in a VLIW Processor" filed June 16, 2000, U.S. Patent Application Serial No. 09/598,567 entitled "Methods and Apparatus for Improved Efficiency in Pipeline Simulation and Emulation" filed June 21, 2000, U.S. Patent Application Serial No. 09/598,564 entitled "Methods and Apparatus for Initiating and Resynchronizing Multi-Cycle SIMD Instructions" filed June 21, 2000, U.S. Patent Application Serial No. 09/598,566 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed June 21, 2000, and U.S. Patent Application Serial No. 09/598,084 entitled "Methods and Apparatus for Establishing Port Priority Functions in a VLIW Processor" filed June 21, 2000, U.S. Patent Application Serial No. 09/599,980 entitled "Methods and Apparatus for Parallel Processing Utilizing a Manifold Array (ManArray) Architecture and Instruction Syntax" filed June 22, 2000, U.S. Patent Application Serial No. ____ entitled "Methods and Apparatus for Scalable Array Processor Interrupt Detection and Response" filed February 23, 2001, U.S. Patent Application Serial No. ____ entitled "Methods and Apparatus for Providing Bit-Reversal and Multicast Functions Utilizing DMA Controller" filed February 23, 2001, as well as, Provisional Application Serial No. 60/113,637 entitled "Methods and Apparatus for Providing Direct Memory Access (DMA) Engine" filed December 23, 1998, Provisional Application Serial No. 60/113,555 entitled "Methods and Apparatus Providing Transfer Control" filed December 23, 1998, Provisional Application Serial No. 60/139,946 entitled "Methods and Apparatus for Data Dependent Address Operations and Efficient Variable Length Code Decoding in a VLIW Processor" filed June 18, 1999, Provisional Application Serial No. 60/140,245 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed June 21, 1999, Provisional Application Serial No. 60/140,163 entitled "Methods and Apparatus for Improved Efficiency in Pipeline Simulation and Emulation" filed June 21, 1999, Provisional Application Serial No. 60/140,162 entitled "Methods and Apparatus for Initiating and Re-Synchronizing Multi-Cycle SIMD Instructions" filed June 21, 1999, Provisional Application Serial No. 60/140,244 entitled "Methods and Apparatus for Providing One-By-One Manifold Array (1x1 ManArray) Program Context Control" filed June 21, 1999, Provisional Application Serial No. 60/140,325 entitled "Methods and Apparatus for Establishing Port Priority Function in a VLIW

Processor" filed June 21, 1999, Provisional Application Serial No. 60/140,425 entitled "Methods and Apparatus for Parallel Processing Utilizing a Manifold Array (ManArray) Architecture and Instruction Syntax" filed June 22, 1999, Provisional Application Serial No. 60/165,337 entitled "Efficient Cosine Transform Implementations on the ManArray Architecture" filed November 12, 1999, and Provisional Application Serial No. 60/171,911 entitled "Methods and Apparatus for DMA Loading of Very Long Instruction Word Memory" filed December 23, 1999, Provisional Application Serial No. 60/184,668 entitled "Methods and Apparatus for Providing Bit-Reversal and Multicast Functions Utilizing DMA Controller" filed February 24, 2000, Provisional Application Serial No. 60/184,529 entitled "Methods and Apparatus for Scalable Array Processor Interrupt Detection and Response" filed February 24, 2000, Provisional Application Serial No. 60/184,560 entitled "Methods and Apparatus for Flexible Strength Coprocessing Interface" filed February 24, 2000, Provisional Application Serial No. 60/203,629 entitled "Methods and Apparatus for Power Control in a Scalable Array of Processor Elements" filed May 12, 2000, and Provisional Application Serial No. 60/241,940 entitled "Methods and Apparatus for Efficient Vocoder Implementations" filed October 20, 2000, and Provisional Application Serial No. 60/251,072 entitled "Methods and Apparatus for Providing Improved Physical Designs and Routing with Reduced Capacitive Power Dissipation" filed December 4, 2000, all of which are assigned to the assignee of the present invention and incorporated by reference herein in their entirety.

A coprocessor interface in accordance with the present invention is obtained by generalizing the debug mechanism and using it to function as a general coprocessor interface. The basic debug mechanism is described first in the context of an exemplary ManArray processor system, and then it is extended to function as a general coprocessor interface for other contexts utilizing other processors.

In a presently preferred embodiment of the present invention, a ManArray 2x2 iVLIW single instruction multiple data stream (SIMD) processor 100 as shown in Fig. 1 may be adapted as described further below for use in conjunction with the present invention. Processor 100 comprises a sequence processor (SP) controller combined with a processing element-0 (PE0) to form an SP/PE0 combined unit 101, as described in further detail in U.S. Patent Application Serial No. 09/169,072 entitled "Methods and Apparatus for Dynamically Merging an Array Controller with an Array Processing Element". Three additional PEs 151, 153, and 155 are also labeled with their matrix positions as shown in parentheses for PE0 (PE00) 101, PE1 (PE01)151, PE2 (PE10) 153, and PE3 (PE11) 155. The SP/PE0 101 contains an instruction fetch (I-fetch) controller 103 to allow the fetching of "short" instruction words (SIW) or abbreviated- instruction words from a B-bit instruction memory

105, where B is determined by the application instruction-abbreviation process to be a reduced number of bits representing ManArray native instructions and/or to contain two or more abbreviated instructions as described in the present invention. If an instruction abbreviation apparatus is not used then B is determined by the SIW format. The fetch controller 103 provides the typical functions needed in a programmable processor, such as a program counter (PC), a branch capability, eventpoint loop operations (see U.S. Provisional Application Serial No. 60/140,245 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed June 21, 1999 for further details), and support for interrupts. It also provides the instruction memory control which could include an instruction cache if needed by an application. In addition, the I-fetch controller 103 controls the dispatch of instruction words and instruction control information to the other PEs in the system by means of a D-bit instruction bus 102. D is determined by the implementation, which for the exemplary ManArray coprocessor D=32-bits. The instruction bus 102 may include additional control signals as needed in an abbreviated-instruction translation apparatus.

In this exemplary system 100, common elements are used throughout to simplify the explanation, though actual implementations are not limited to this restriction. For example, the execution units 131 in the combined SP/PE0 101 can be separated into a set of execution units optimized for the control function; for example, fixed point execution units in the SP, and the PE0 as well as the other PEs can be optimized for a floating point application. For the purposes of this description, it is assumed that the execution units 131 are of the same type in the SP/PE0 and the PEs. In a similar manner, SP/PE0 and the other PEs use a five instruction slot iVLIW architecture which contains a VLIW instruction memory (VIM) 109 and an instruction decode and VIM controller functional unit 107 which receives instructions as dispatched from the SP/PE0's I-fetch unit 103 and generates VIM addresses and control signals 108 required to access the iVLIWs stored in the VIM. Referenced instruction types are identified by the letters SLAMD in VIM 109, where the letters are matched up with instruction types as follows: Store (S), Load (L), ALU (A), MAU (M), and DSU (D).

The basic concept of loading the iVLIWs is described in further detail in U.S. Patent Application Serial No. 09/187,539 entitled "Methods and Apparatus for Efficient Synchronous MIMD Operations with iVLIW PE-to-PE Communication". Also contained in the SP/PE0 and the other PEs is a common PE configurable register file 127 which is described in further detail in U.S. Patent Application Serial No. 09/169,255 entitled "Method and Apparatus for Dynamic Instruction Controlled Reconfiguration Register File with Extended Precision". Due to the combined nature of the SP/PE0, the data memory interface

controller 125 must handle the data processing needs of both the SP controller, with SP data in memory 121, and PE0, with PE0 data in memory 123. The SP/PE0 controller 125 also is the controlling point of the data that is sent over the 32-bit or 64-bit broadcast data bus 126. The other PEs, 151, 153, and 155 contain common physical data memory units 123', 123'', and 123''' though the data stored in them is generally different as required by the local processing done on each PE. The interface to these PE data memories is also a common design in PEs 1, 2, and 3 and indicated by PE local memory and data bus interface logic 157, 157' and 157''. Interconnecting the PEs for data transfer communications is the cluster switch 171 various aspects of which are described in greater detail in U.S. Patent Application Serial No. 08/885,310 entitled "Manifold Array Processor", now U.S. Patent No. 6,023,753, and U.S. Patent Application Serial No. 09/169,256 entitled "Methods and Apparatus for Manifold Array Processing", and U.S. Patent Application Serial No. 09/169,256 entitled "Methods and Apparatus for ManArray PE-to-PE Switch Control". The interface to a host processor, other peripheral devices, and/or external memory can be done in many ways. For completeness, a primary interface mechanism is contained in a direct memory access (DMA) control unit 181 that provides a scalable ManArray data bus 183 that connects to devices and interface units external to the ManArray core. The DMA control unit 181 provides the data flow and bus arbitration mechanisms needed for these external devices to interface to the ManArray core memories via the multiplexed bus interface represented by line 185. A high level view of a ManArray control bus (MCB) 191 is also shown in Fig. 1.

Debug Operation

Fig. 2 shows an exemplary system 200. In system 200, a coprocessor 210 which may suitably be a ManArray DSP 2x2 (2x2) has debug and control registers 220 which may be accessed both by coprocessor instructions and by bus masters residing on ManArray control bus (MCB) 230. In this exemplary system, the coprocessor instructions used are the load from special-purpose register (LSRP) and store to special-purpose register (SSPR) instructions. These instructions may be used to access all of the debug and control registers 220 which are visible to MCB master devices. A ManArray data bus (MDB) 240 is also shown, connecting coprocessor local memories within the boundary of 210 via a DMA controller to other memory or input/output (I/O) devices residing on the MDB. A "test module" 250 is shown which acts as a bus master on the MCB and which is capable of initiating read and write cycles to the coprocessor control and debug registers 220. The test module 250 has read/write access to the coprocessor's instruction memory. This test module may represent a host control processor, or other interface logic which allows a standard debug path, such as JTAG, to connect to the MCB 230 and issue read and write cycles.

There is also provided a mechanism by which the test module may initiate a debug interrupt signal to the coprocessor either by writing to a particular MCB address or by configuring certain registers to assert the debug interrupt signal when the coprocessor reaches a specified program state, either an instruction or data address, for example. This latter mechanism is preferably programmed utilizing a set of event point registers described in U.S. Patent Application Serial No. 09/598,566 and U.S. Provisional Application Serial No. 60/140,245 both entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" and filed June 21, 2000 and June 21, 1999, respectively, both of which are incorporated by reference herein in their entirety.

Debug interface and usage are described below:

The test module 250 of Fig. 2 initiates a debug interrupt signal to the coprocessor. This may be done as described above, either by writing to a particular MCB address or by configuring an event point register to trigger the debug interrupt when a coprocessor program state instruction address or data address, for example, is reached.

The coprocessor responds to the debug interrupt by saving the current program state (all essential registers) and fetching an instruction memory address (called an "interrupt vector") from a region of instruction memory 310 seen in Fig. 3 called the interrupt vector table (IVT) 340. The debug interrupt vector 320 in an exemplary system is shown located at instruction memory address 0x0008, 322. The value stored at 0x0008 is loaded via the instruction bus 325 through a multiplexer 317 to the program counter register (PC) 360. The PC then supplies the next address from which to fetch an instruction via the instruction address bus 365.

If the address stored in the debug interrupt vector location 322 is that of an instruction outside of the IVT, then the instruction at this address is fetched via instruction bus path 327 through the multiplexer 337 to the instruction decode register (IDR) 350. It is subsequently decoded and executed and further instructions are processed starting from that address. In this fashion, a debug interrupt service routine may be located somewhere in the instruction memory, and this may be used to enter a debug monitor program and thereby interact with the test module. This is one form of debug initiation for which the present invention is suitable. A second form of debug initiation is also provided in the following event.

If the address stored in the debug interrupt vector is a particular value, 0x0004 in the exemplary system, then the interrupt processing logic operates in a unique fashion. This address, stored in memory location at 320 of Fig. 3, may be configured to cause the interrupt logic to fetch its next instruction from a debug instruction register (DBIR) rather than from the instruction memory address 0x0004. Fig. 4 shows an exemplary DBIR 400. This register

400 is visible to bus masters on the MCB, and in particular to the test module 250 of Fig. 2. A second register called the debug status register (DBSTAT) 500 shown in Fig. 5, controls the behavior of the coprocessor when it fetches from the DBIR in response to a fetch from address 0x0004. A bit in DBSTAT register 530, called the "debug instruction present" (DBIP) bit is used to indicate whether or not an instruction may be fetched.

If the DBIP bit of DBSTAT is zero, then when the coprocessor attempts to fetch an instruction from address 0x0004 in the DBIR register, it stalls, preventing updates to any processor state. If the test module then writes an instruction to the DBIR, the DBIP bit of DBSTAT is set to 1. This causes the coprocessor to fetch this instruction from the DBIR, and the DBIP bit is cleared, thereby causing the coprocessor to stall until the next instruction is written to the DBIR. In this manner, the test module may gain control of the coprocessor by feeding it instructions one at a time. Whenever the DSP is stalled waiting for an instruction to be placed into the DBIR, the debug stall bit (DBSTALL) 520 is set to 1 in the DBSTAT register. This bit may be used by the test module or other control processor having MCB access to indicate when the coprocessor is in a stalled state waiting for a debug instruction.

Two additional control bits are provided in a DSP control register (DSPCTL) shown in Fig. 6 600. The debug instruction register enable bit (DBIREN) 620, when set, causes any instruction fetch from address 0x0004 to be redirected to fetch from the DBIR as described above. If this bit is cleared to zero, then a fetch from 0x0004 behaves as if it were any other instruction and the contents of this address are sent to the IDR 350, of Fig. 3 for decode and subsequent execution. This allows the IVT address to be optionally used for a normal interrupt vector. The LOCKPC bit of the DSPCTL register is used to prevent the PC from incrementing or updating at all after fetching and executing instructions. Since the DSPCTL register is accessible by MCB masters, this bit is also accessible by the test module. In normal operation, the automatic incrementing of the PC after instruction fetch is inhibited while fetching instructions from the DBIR. If a branch instruction is executed, then the branch address is loaded into the PC and subsequent instructions are fetched with automatic PC incrementing re-enabled. When the LOCKPC bit is set, even branch type instructions will not affect the PC value. This allows program sequences to be executed through the DBIR port in such a manner as to filter out branches.

An alternative approach for controlling PC auto-incrementing is through the use of a second MCB address for MCB writes to the DBIR. When the first address is used for writing instructions to the DBIR, the PC only updates when a branch instruction is executed. When the second address is used, the PC is locked or prevented from updating for all instructions written and branches are ignored.

By using the debug mechanisms outlined above, a debug interrupt can be made to cause the coprocessor to stall, waiting for instructions to be sent to it through the DBIR. When this occurs, the test module may issue instructions through the DBIR which, by executing, dump the processor state out to an external memory or the test module itself for external storage. Two additional registers are provided which allow coprocessor state to be saved without corrupting it. A debug data-out register (DBDOUT) 700 seen in Fig. 7 and a debug data-in register (DBDIN) 800 seen in Fig. 8 are used for this purpose. When the coprocessor writes to the DBDOUT register, a bit in the DBSTAT register 540 seen in Fig. 5 is set. This bit called the debug data output buffer full (DBDOBF) bit indicates that the DBDOUT register contains data written by the coprocessor. A read of the DBDOUT register by the test module will cause this bit to be automatically cleared. To allow the coprocessor to read data from the test module (used typically for restoring state, or debugger communication), the DBDIN register is used. An MCB write to the DBDIN register causes the debug data input buffer full (DBDIBF) bit 550 of Fig. 5 to be set in the DBSTAT register, indicating that there is data available. This bit is cleared when the coprocessor performs a read of the DBDIN register. For each of the two data registers, DBDOUT and DBDIN, secondary addresses (MCB and LSPR/SSPR) for data reads are provided which allow the register to be read without clearing the DBDOBF or DBDIBF bits.

Once the coprocessor stalls waiting for an instruction, it is possible for the test module to access the coprocessor's instruction memory. A region of this memory may then be copied (read and stored) to an external data store and replaced with a debug monitor program. By writing a branch instruction (e.g. JMPD) to the DBIR, the test module can then direct execution to the "inserted" debug monitor. This monitor code may be used to dump selected state information, such as register and memory contents, to the DBDOUT register for reading by the test module. When the debug function is complete, the last instruction causes a branch to the DBIR address (0x0004). The coprocessor then stalls waiting for the next instruction and the application's instructions are restored to the instruction memory by the test module. When the last debug module has been executed, a "return-from –interrupt" (RETI) instruction is written to the DBIR by the test module. The coprocessor fetches this instruction causing the application to return to its pre-interrupt state and resume execution. During execution of the debug monitor code, the DBDIN register can be used by the test module to pass data or commands to the "inserted" monitor code.

To single-step the processor, the debug interrupt request bit in the interrupt request register (IRR) may be set explicitly by a coprocessor instruction. When the RETI is executed, the coprocessor will return to executing the application code for one instruction

before taking the pending debug interrupt. This allows one instruction to be fetched and executed at a time.

Coprocessor Operation

With this understanding of the debug event sequence and debug registers, it is next explained how the debug interface may be generalized to provide a coprocessor interface with varying levels of coupling.

The debug interface described herein may advantageously have the following general characteristics. A segment of instruction memory (the IVT 340 of Fig. 3 in this case) has at least two modes of access. The first mode is such that the data from an address is treated as a vector or branch target and placed into the PC 360, and occurs when responding to an interrupt. The second mode is when the data from an address is treated as an instruction to be placed in the IDR 330 for subsequent decode and execution.

One or more addresses within this special segment of memory have "shadow" instruction registers or memory buffers associated with them. That is, these addresses may access one physical memory location during one access mode (e.g. interrupt vector fetch) and the second "shadow" location for another access mode (e.g instruction fetch). For the case of debug, the IVT address 0x0004 320 is shadowed by the DBIR register 330 in the preceeding discussion. These shadow registers may alternatively be replaced by memory buffers which operate is first-in-first-out (FIFO) queues when an external device writes instructions to them and when the coprocessor fetches instructions from them.

There are control bits associated with each of the special addresses that are shadowed. One bit controls whether the shadow function (debug instruction execution via DBIR in the discussion above) is enabled or disabled, a second is used to indicate when an instruction is available to the coprocessor from the shadow instruction register or buffer (e.g. DBIP 530 in Fig. 5), and a third is used to indicate to an external device when the coprocessor is stalled waiting for an instruction (e.g. DBSTALL 520). An optional fourth bit may be used to control locking of the PC thereby disallowing branches to redirect program execution from the shadow instruction address.

Bus addresses are provided which allow an external bus master to write instructions to the shadow instruction registers. Each shadow instruction register has a pair of addresses. Writing instructions to the first address allows branch instructions to cause the PC to be updated (i.e. the branch to be executed), and writing instructions to the second address does not allow the PC to be updated for branch instructions. This is analogous to the LOCKPC bit of the DSPCTL register described above.

Interface registers are provided for inter-processor communication, along with

associated control and status bits for indicating when they are read/written (DBDIN 800 of Fig. 8 and DBDOUT 700 of Fig. 7 along with the DBDOBF 540 of Fig 5 and DBDIBF 550 bits of the DBSTAT register in the above discussion). In the exemplary system, these registers may be accessed using LSPR and SSPR instructions by the DSP coprocessor and by read/write accesses on the MCB by the test module or host control processor.

The characteristics listed above may be used to implement a more general coprocessor interface which may be used in tightly coupled, loosely coupled and firmly coupled coprocessing systems as well as for debug operations. A given implementation might advantageously allow the sharing of a single interface for both coprocessing and debug purposes such as that described in the preceeding section, or independent interfaces (separate shadowed instruction register or memory buffer addresses) for the coprocessing and debug functions. An implementation might also use multiple instances of the interface to allow multiple external processing devices to send instructions to the coprocessor. The following sections describe various types of applications consistent with the teachings of the present invention.

Tightly-Coupled Co-processing Systems

For a tightly coupled coprocessing system, the coprocessor does not have its own independent instruction fetch logic or PC. Fig. 9 shows a high-level view of a representative system 900. A host control processor (HCP) 910 , a coprocessor 920 and a system memory 930 are shown connected by a system data bus (SDB) 940, a coprocessor data bus (CDB) 950, a coprocessor instruction bus 929 and coprocessor control signals 928. A memory bus interface unit (MBIU) 912 provides a data path for the HCB to read/write data from the data cache 964 or access the system data bus SDB or CDB 950. An instruction fetch unit (IFU) 914 controls HCP instruction fetch sequence and the cache and bus interface control unit (CBICU) manages instruction cache 962 accesses, cache line refills via the SDB 940 from system memory 930, and dispatches instructions to the coprocessor via the coprocessor instruction bus 929 and coprocessor control signals 928.

The coprocessor contains at least one coprocessor instruction register (COIN) 923, a coprocessor status and control register (COSTAT) 924, coprocessor data bus interface logic 925 that allows the coprocessor to access host registers via LSPR- or SSPR-type instructions and the HCB to access coprocessor registers, a coprocessor control register (DSPCTL) 927, a DMA controller 926, and local data memories. The DSPCTL register is used to control high level coprocessor functions such as RESET, and the LOCKPC function as described in the preceeding sections.

During operation, the HCP fetches instructions via the CBICU. Instructions are

dispatched in parallel to the COIN register 923 via the coprocessor instruction bus 929 and also to the HCB via its instruction fetch bus 918. An alternate, and preferred approach is to separate coprocessor instructions from HCP instructions using different memory regions, and designing the CBICU 960 to issue instructions only to the coprocessor when executing in its memory region. The HCP in the exemplary embodiment must always be aware of branch instructions, therefore, unless this capability is provided in the CBICU, it is necessary to always issue instructions to the HCP, even if they are treated as no-operation (NOP) instructions. This arrangement has the advantage of saving power when the coprocessor is not in use. When an instruction is received at the COIN register, a bit in the COSTAT register, IPRES (analogous to the DBIP bit 530 of Fig. 5) is set to indicate an instruction is available. For tightly-coupled operation, a bit COREADY (directly analogous to the DBSTALL bit 520 of Fig. 5) is used to provide flow control to the HCP CBICU instruction dispatch logic. It is set whenever the coprocessor is able to receive an instruction in the COIN register and cleared otherwise. In order to allow instruction processing to proceed with minimum delay, for tightly-coupled systems, the COREADY bit may be generated with combinational logic based upon the IPRES bit and the state of the coprocessor pipeline so that the coprocessor can receive an instruction on each cycle. Coprocessor instructions fetched by the HCP are ignored by it, but they are processed immediately by the coprocessor via the coprocessing interface. The mechanism used may be shared with debug logic also. In this manner, the HCB controls the instruction sequence while the coprocessor executes instructions that are not native to the HCP architecture. HCP-Coprocessor register-to-register transfers are provided via the CDB. The coprocessor provides special instructions for this purpose (e.g. LSPR and SSPR) and the HCP either provides special instructions or maps certain parts of its data address space so that accesses to those regions are converted to data accesses to the coprocessor registers by means of the CDB.

Tightly-coupled processing is characterized by the fact that while the coprocessor is executing instructions, the HCP is not executing instructions other than branch-type instructions which are ignored by the coprocessor. Selected condition information from coprocessor execution is provided by means of the coprocessor control signals 928. The same hand-shake operation by means of the IPRES (DBIP) bit and the COREADY (DBSTALL) bit are carried.

Loosely-Coupled Coprocessing

Fig. 10 shows a system 1000 very similar to system 900 of Fig. 9. In this representative system 1000, the the coprocessor is equipped with a program counter register (PC) 1070 and local instruction memory 1020. A further modification is that the CDB 950 of

Fig. 9 has been changed to a more general system control bus 1050. This bus 1050 provides access to the same registers as with the tightly-coupled system but with longer latency and somewhat lower hardware cost, because it is not assumed that the HCP will supply instructions to the processor as frequently. While the system shown in Fig. 10 has an SCB 1050, depending on the performance requirements, the SDB 1040 might be the only bus required for the system, in which case all inter-processor communication occurs on a single bus.

It system 1000, the HCP initiates coprocessor execution at procedure or program granularity rather than at instruction granularity. DMA controller 1026 may be programmed to load data and instruction memories. The HCP can write an instruction to COIN 1023 via the SCB 1050. This instruction would typically be a direct branch (JMPD) or call (CALLD) instruction. In order to allow a subroutine to return to wait at the COIN register for the next function, the subroutine must branch back to the COIN register address, (such as 0x0004 for the debug case, though this address might be another address within the IVT for a coprocessor interface. Normally, a CALL type instruction returns to the address following the instruction itself. If the CALL type instruction is read from the COIN register, a return would cause the next instruction fetch to occur at the address following the address of the COIN register. This could be resolved by saving the address of the CALL instruction itself for those cases in which the instruction comes from the COIN register. In the exemplary system, the return address is saved in a register called the user-link register (ULR). This register is programmer visible and may be modified by load-type or copy instructions. The behavior of the coprocessor interface and signals is the same as for the debug or tightly-coupled cases, including the use of the IPRES and COREADY bits.

Loosely-coupled processing is characterized by having the coprocessor execute entire functions or programs before returning to look for further instructions from the host processor. Data may be communicated between the processors through registers similar to the DBDOUT and DBDIN registers of Figs. 7 and 8, respectively.

Firmly-Coupled Coprocessing

Fig. 11 shows a representative system 1100 which may be considered a "firmly-coupled" coprocessing system. Coprocessor 1120 in this implementation contains a PC 1170 just as with the loosely-coupled system 1000 of Fig. 10. It also has its own local instruction memory and the same coprocessor interface registers COIN 1123, COSTAT 1124, and DSPCTL 1127 as the other system described above. System 1100 also includes a coprocessor instruction bus 1129 and interface control signals 1128 that are used in tightly-coupled systems, but absent from loosely-coupled systems. One idea behind this class of

system is that it has a limited level of autonomy. That is, while it has a PC, the instruction memory may be fairly small and processing may be focused on particular types of instruction sequences, for example, the processing of inner loops of functions. It also may receive instructions directly from the HCP 1114 by way of a cache and bus interface control unit (CBICU) 1160. The COIN register 1123 may also be extended to become a first-in-first-out (FIFO) queue for instructions. Providing a FIFO buffer for capturing instructions from the HCP allows the clock rates of the HCP and coprocessor to differ while maintaining a clean interface at the FIFO. If the HCP is capable of a higher clock rate, providing an instruction buffer allows it to dispatch multiple instructions to a coprocessor at full speed rather than being interlocked to the processor. The control interface for the FIFO retains the same IPRES and COREADY bits which in this case indicate "FIFO not empty" (coprocessor instructions available) and "FIFO not full" (room for more instructions from HCP).

In one exemplary system, such as a ManArray indirect-VLIW DSP, there are two classes of instruction memories. One contains "short" instructions, typically 32-bits in the current embodiment, and the other contains very-long-instruction words (VLIWs). The PC is used to access the short instruction memory. A certain type of (short) instruction, called an execute-VLIW (XV) instruction, may be used to indirectly reference a VLIW instruction from the VLIW instruction memory. When decoded, the XV instruction causes a VLIW to be accessed and executed. A VLIW consists of multiple instructions which are executed in parallel. Another type of "short" instruction is the "load-VLIW" (LV) instruction which is used to load the VLIW instruction memories. The LV instruction is followed by one or more instructions which are not executed immediately, but are rather placed into a VLIW instruction memory address specified by the LV instruction. A firmly coupled coprocessing system utilizing a ManArray DSP might then provide a small short instruction memory used for loading and executing VLIWs and a coprocessor interface through which the HCP writes branch instructions to the COIN register to initiate VLIW instruction processing.

The ManArray DMA controller of the exemplary system also has characteristics that allow it to combine with the coprocessor to allow a greater level of autonomy without large instruction memory cost and low overhead on the HCP. The DMA controller is able to fetch its own instructions from the coprocessor local memory. Based on these instructions, it can then load the coprocessor instruction memory, data memories, and then send a message to the COIN register which is a branch instruction to the program code entry point. Alternatively, completion of the DMA transfers may be configured to signal the HCP that the coprocessor program is ready for execution, and the HCP can issue a branch instruction to the COIN register. If additional DMA instructions are included in the transfers associated with the first

task, the DMA instruction fetch logic is able to branch to the next set of DMA instructions and so load the next task while the first task is executing on the coprocessor. HCP involvement in this process is then reduced to managing groups of DMA instructions, each of which causes the execution of a coprocessor task or function.

Client-Server Coprocessing

Given the flexibility of the coprocessing interface described in the preceeding sections, it is possible to extend the capability further to allow multiple coprocessing interfaces for a single coprocessor. These interfaces might be either loosely-coupled or firmly-coupled depending on the application. By providing multiple interfaces (COIN registers or FIFOs), it is possible to construct systems in which a single powerful coprocessor is able to act as a "server" for multiple "client" HCPs. Each "client" has its own coprocessor instruction FIFO interface which is serviced in turn by the coprocessor. This multi-queue–single-server model may be useful for high-performance compute server processing cores which can service multiple control processor clients, such as "client-server-on-a-core" or in other terms a client-server system on chip (SOC or CSOC).

In the discussion which follows, the following terminology is used: the server processor or DSP is called the SCOP. The client control processors are designated CCPs. In this embodiment of the present invention, a single instruction port on the SCOP is used for each CCP, which is essentially an instruction FIFO together with special control logic to allow the SCOP to stall when the FIFO is empty, and in addition allow it to operate in either a tightly coupled (locked PC) mode, or a firmly coupled (branch-capable) mode. Instruction requests are posted to the queues by the CCPs and the SCOP processes requests according to a scheduling program which it executes after each request is serviced (for SCOPs with PCs). This type of operation requires the CCPs to manage the setup of data I/O for the SCOP, and synchronization.

While the present invention is disclosed in the context of a presently preferred embodiment, it will be recognized that a wide variety of implementations may be employed by persons of ordinary skill in the art consistent with the above discussion and the claims which follow below.

I claim:

1. A method for providing flexible coupling between a coprocessor and a control processor comprising the steps of:

establishing an external instruction interface by dynamically loading program code and executing the code caused by a control processor initiating the loading process through the external instruction interface; and

unloading the program code when specified by the control processor.

2. The method of claim 1 wherein the program code is debug monitor code.

3. The method of claim 1 wherein the program code is coprocessor function code.

4. The method of claim 1 further comprising the step of utilizing the external instruction interface to flexibly provide a coprocessor interface which supports tightly coupled, loosely coupled and firmly coupled operation of the coprocessor and the control processor.

5. The method of claim 4 wherein the coprocessor is a manifold array (ManArray) processor and tightly coupled operation specifies that the coprocessor receives its instruction from the control processor.

6. The method of claim 5 further comprising the steps of:

the coprocessor accepting an instruction from the control processor which is not part of the coprocessor's instruction set;

the coprocessor executing the instruction from the control processor; and

the control processor not processing further instructions of its own until the coprocessor has completed execution of its instruction.

7. The method of claim 4 wherein the coprocessor is a manifold array (ManArray) processor and loosely coupled operation specifies that the control processor dispatches a signal or message to the coprocessor causing it to execute special subroutines in parallel with the control processor's own execution.

8. The method of claim 7 further comprising the steps of:

the coprocessor executing a special subroutine; and

upon completion of the special subroutine, waiting for another subroutine to execute.

9. The method of claim 4 wherein the coprocessor is a manifold array (ManArray) processor and firmly coupled operation specifies that the coprocessor can optionally fetch instructions from an instruction first-in-first-out (FIFO) buffer.

10. The method of claim 9 further comprising the steps of:

the control processor directing the coprocessor to execute a concurrent subroutine by

placing a call or branch-type instruction in the instruction FIFO buffer;

upon completion of the concurrent subroutine, the coprocessor branching to the address in the instruction FIFO buffer;

if more instructions are available, then executing those instructions; and

if more instructions are not available, then stalling the coprocessor and waiting for further instructions.

11. The method of claim 6 wherein a further coprocessor is employed, which does not have a program counter (PC), and all instruction fetches are from the external instruction interface.

12. A method for providing a flexible coupling mechanism between two or more processors in a multiprocessor system that must communicate to accomplish a task, the method comprising the steps of:

initiating a debug interrupt on a target digital signal processor (DSP) core which comprises one of the two processors that must communicate utilizing a test module residing on a control bus (MCB) in response to an external debugger program; and

fetching a debug vector stored in an interrupt vector table.

13. The method of claim 12 wherein the debug vector contains the address of a debug instruction register (DBIR) and further comprising the step of:

stalling a processor seeking to read the DBIR when a debug instruction present (DBIP) bit is clear.

14. The method of claim 12 wherein the debug vector contains the address of a debug instruction register (DBIR) and further comprising the step of:

returning the value read to an instruction decode register (IDR) when a debug instruction present (DBIP) bit is set.

15. The method of claim 13 further comprising the step of:

reading the DBIR value in response to a sequence processor (SP) load special purpose register (LSPR) instruction returns the DBIR value to an SP IDR register with no effect on the DBIP bit.

16. The method of claim 13 further comprising the step of:

writing in response to an MCB access to update the DBIR with no effect on the DBIP bit.

17. The method of claim 13 further comprising the step of:

posting a status bit (DBSTALL) to a DBSTAT register to indicate a stall in effect.

18. The method of claim 17 wherein the DBSTAT register includes a status bit which is debug data in buffer full (DBDIF) bit, a read only bit, and further comprising the

step of:

setting the DBDIF bit when an MCB device writes to a DBDIN register; and

clearing the DBDIF bit when a sequence processor (SP) reads from the DBDIN register.

19. The method of claim 17 wherein the DBSTAT register includes a status bit which is a debug data out buffer full (DBDOBF) bit, a read only bit, and further comprising the steps of:

setting the DBDOBF bit when a sequence processor (SP) writes to a DBOUT register; and

clearing the DBDOBF bit when the MCB reads from the DBDOUT register's MCB address.

20. The method of claim 17 wherein the DBSTAT register includes a status bit which is the DBIP bit, and the method further comprises:

setting the DBIP bit when an MCB device writes to the DBIR via its MCB address or when a sequence processor (SP) writes to it via an SPR address; and

clearing the DBIP bit when an instruction fetch accesses the DBIR address with the DBIP bit set.

21. The method of claim 17 wherein the DBSTAT register includes the DBSTALL bit, a read only bit, and the method further comprises the steps of:

setting the DBSTALL bit when a sequence processor attempts to fetch an instruction from the DBIR when the DBIP is cleared.

22. The method of claim 17 further comprising the step of:

detecting a debug stall bit set utilizing the test module;

reading a section of instruction memory using MCB accesses;

saving the read section of memory in an external location; and

injecting debug monitor code into this read region of memory.

23. The method of claim 22 further comprising the step of:

writing a jump direct (JMPD) branch instruction in the DBIR utilizing the test module.

24. The method of claim 23 further comprising the step of:

setting the DBIP bit of the DBSTAT register to indicate that an instruction is present in the DBIR.

25. The method of claim 24 further comprising the steps of:

retrieving this instruction by the coprocessor; and

executing it by the coprocessor.

26. The method of claim 22 further comprising the steps of:

executing the monitor code; and

dumping the processor state to the DBDOUT register to transmit the data to the test module.

27. A method for providing a flexible coupling mechanism between two or more processors in a multiprocessor system that must communicate to accomplish a task, the method comprising the steps of:

storing an instruction present (IPRES) bit in a DBSTAT register; and

storing an aliased address in an interrupt vector table to provide the address of an external instruction source register which is used as a coprocessor instruction port.

28. The method of claim 27 further comprising the steps of:

modifying a manifold array (ManArray) sequence processor (SP) to act as a coprocessor which is one of the two or more processors; and

causing an execution pipe to stall and not advance when said SP tries to fetch an instruction from said instruction port when the IPRES bit is cleared.

29. The method of claim 27 further comprising the steps of:

modifying a manifold array (ManArray) sequence processor (SP) to act as a coprocessor; and

causing said SP to fetch an instruction from said instruction port and to execute it when the IPRES bit is set.

30. The method of claim 29 further comprising the steps of:

determining if the instruction is a branch-type instruction and a LOCKPC bit is not set (0); and

if the determinations is made, resuming fetching from the branch target.

31. The method of claim 29 further comprising the steps of:

determining if the instruction is a branch-type instruction and a LOCKPC bit is set (1); and

if the determination is made then the branch-type instruction would be treated as an NOP, except for a CALL instruction which load a URL.

32. An interface source system providing at least two paths to the instruction decode register of a coprocessor which is part of a system having two or more processors including the coprocessor that communicates to accomplish some task, the instruction source system comprising:

an instruction port register;

an instruction memory;

an interrupt vector table (IVT) stored in the instruction memory, said IVT storing an external instruction vector containing the address of the instruction port register;

an instruction decode register; and

a program counter register, wherein the instruction decode register can be directly accessed from the instruction memory containing the IVT or indirectly the instruction port register.

33. The instruction source system of claim 32 wherein the instruction port register is a debug instruction register (DBIR) having a debug instruction present (DBIP) bit, and the system further comprises a sequence processor (SP) instruction fetch unit which returns the value read to the instruction decode register when the DBIP bit is set.

34. The instruction source system of claim 32 wherein the instruction port register is a control processor interface register having an instruction present bit (IPRES), and the system further comprises a sequence processor (SP) instruction fetch unit which returns the value read to the instruction decode register when the IPRES bit is set.

35. Apparatus for providing a single coprocessor to act as a server (SCOP) to support a number of client control processors (CCPs) wherein each client has its own coprocessor instruction FIFO buffer interface which is serviced in turn by the single coprocessor acting as the server, the apparatus comprising:

an instruction port on the SCOP for each CCP, which is an instruction FIFO buffer together with control logic to allow the SCOP to stall when the instruction FIFO buffer is empty, and in addition allow it to operate in either a tightly coupled mode or a firmly coupled mode;

a plurality of queues in the SCOP to which the CCPs port requests are posted; and

a scheduling program for the SCOP which controls the SCOP's processing of the posted requests.

FIG. 1

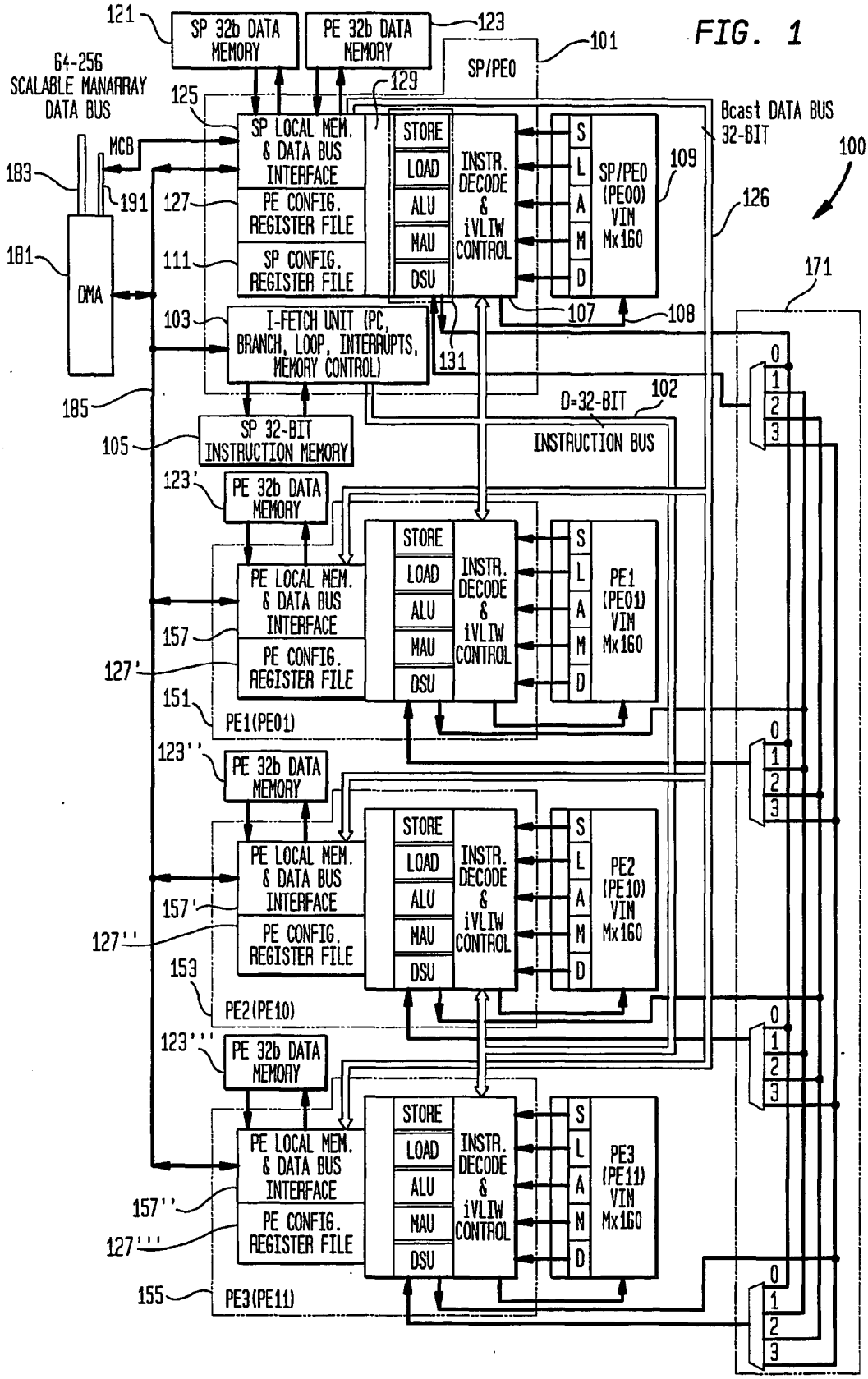

121
SP 32b DATA MEMORY
PE 32b DATA MEMORY
123
101
SP/PE0
64-256 SCALABLE MANARRAY DATA BUS
125
129
Bcast DATA BUS 32-BIT
100
SP LOCAL MEM. & DATA BUS INTERFACE
STORE
LOAD
ALU
MAU
DSU
INSTR. DECODE & iVLIW CONTROL
S
L
A
M
D
SP/PE0 (PE00) VIM Mx160
109
MCB
183
191
127
PE CONFIG. REGISTER FILE
126
181
DMA
111
SP CONFIG. REGISTER FILE
171
103
I-FETCH UNIT (PC, BRANCH, LOOP, INTERRUPTS, MEMORY CONTROL)
107
108
131
185
D=32-BIT
102
105
SP 32-BIT INSTRUCTION MEMORY
INSTRUCTION BUS
123'
PE 32b DATA MEMORY
PE LOCAL MEM. & DATA BUS INTERFACE
157
127'
PE CONFIG. REGISTER FILE
PE1 (PE01) VIM Mx160
151
PE1(PE01)
123''
PE 32b DATA MEMORY
PE LOCAL MEM. & DATA BUS INTERFACE
157'
127''
PE CONFIG. REGISTER FILE
PE2 (PE10) VIM Mx160
153
PE2(PE10)
123'''
PE 32b DATA MEMORY
PE LOCAL MEM. & DATA BUS INTERFACE
157''
127'''
PE CONFIG. REGISTER FILE
PE3 (PE11) VIM Mx160
155
PE3(PE11)
0
1
2
3

FIG. 2

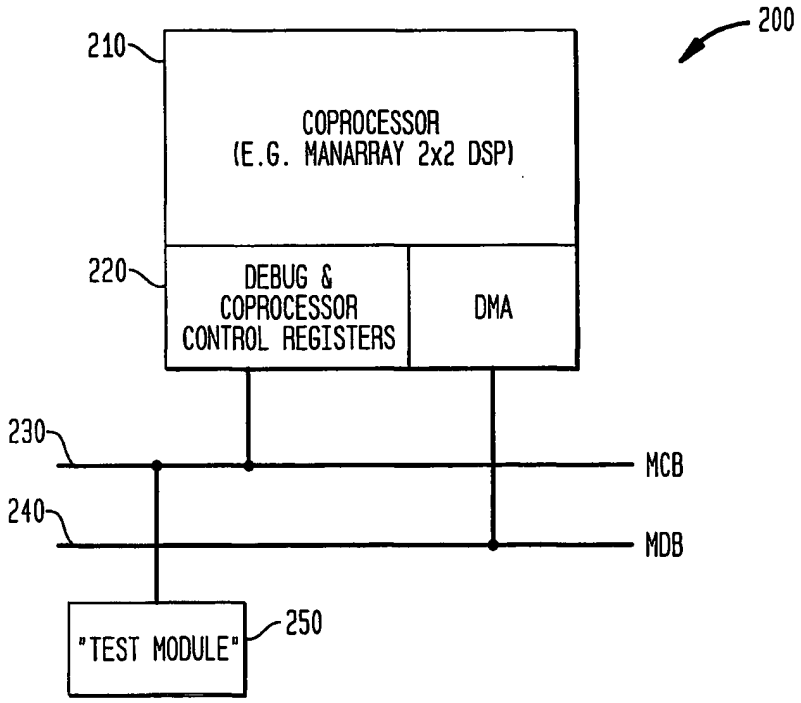

200
210
COPROCESSOR
(E.G. MANARRAY 2x2 DSP)
220
DEBUG &
COPROCESSOR
CONTROL REGISTERS
DMA
230
MCB
240
MDB
"TEST MODULE"
250

FIG. 3

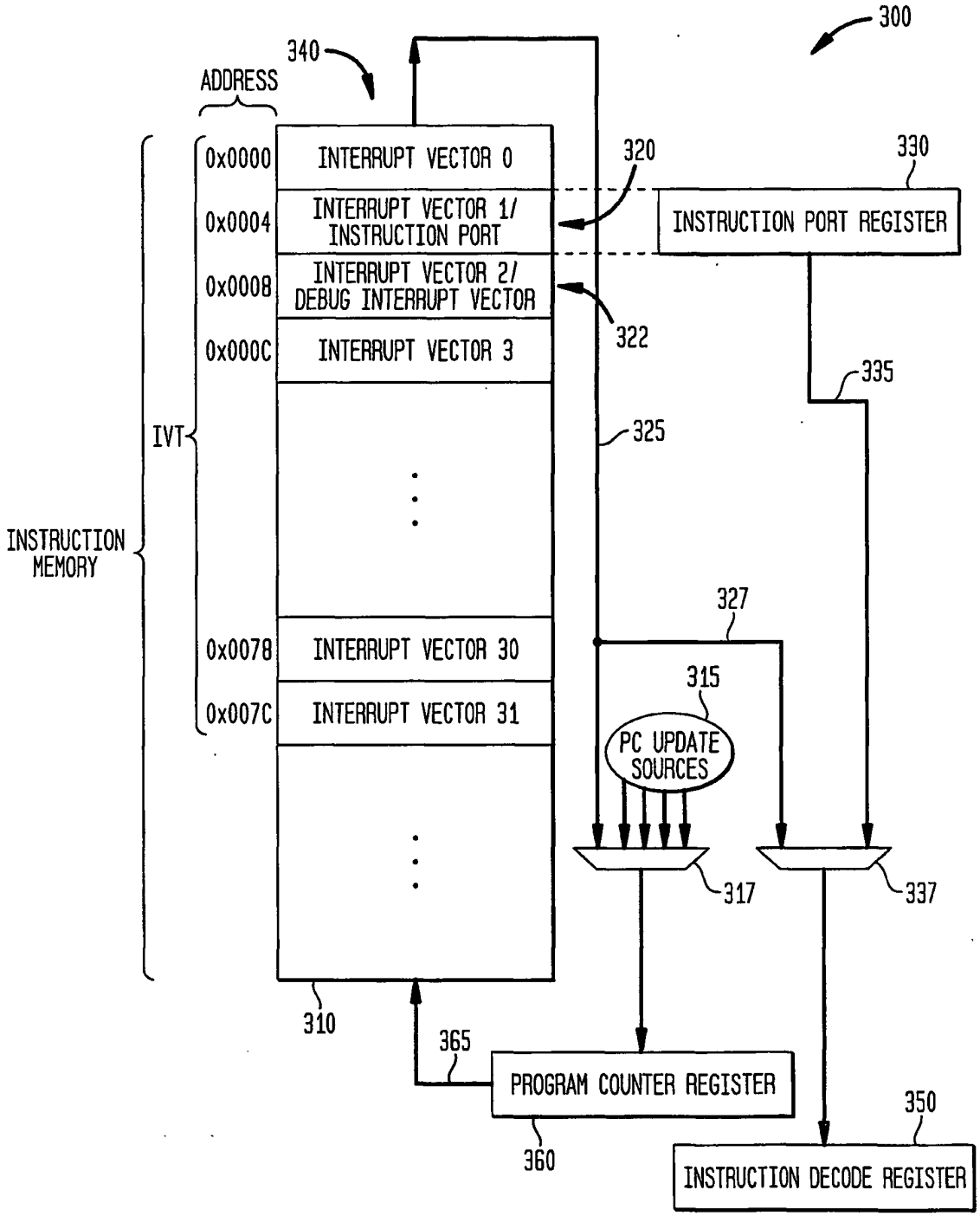

300
340
ADDRESS
0x0000
INTERRUPT VECTOR 0
0x0004
INTERRUPT VECTOR 1/
INSTRUCTION PORT
0x0008
INTERRUPT VECTOR 2/
DEBUG INTERRUPT VECTOR
0x000C
INTERRUPT VECTOR 3
0x0078
INTERRUPT VECTOR 30
0x007C
INTERRUPT VECTOR 31
IVT
INSTRUCTION
MEMORY
320
330
INSTRUCTION PORT REGISTER
322
335
325
327
315
PC UPDATE
SOURCES
317
337
310
365
PROGRAM COUNTER REGISTER
360
350
INSTRUCTION DECODE REGISTER

FIG. 4

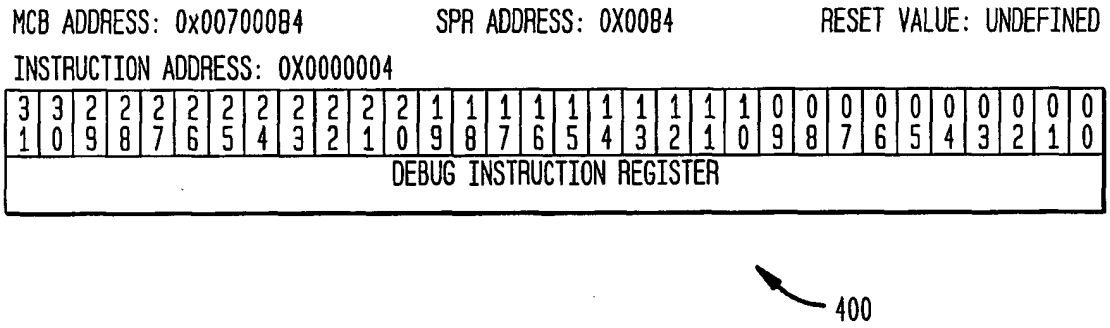

MCB ADDRESS: 0x00700084
SPR ADDRESS: 0X0084
RESET VALUE: UNDEFINED
INSTRUCTION ADDRESS: 0X0000004
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
DEBUG INSTRUCTION REGISTER
400

FIG. 5

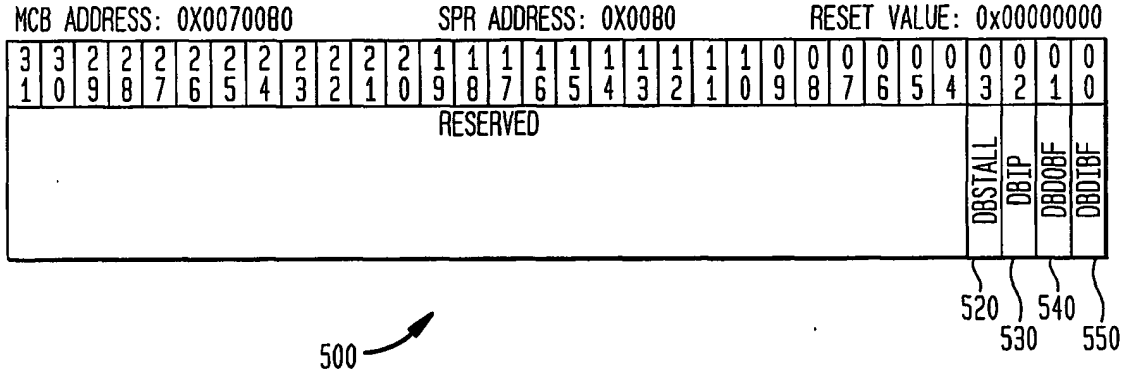

MCB ADDRESS: 0X0070080
SPR ADDRESS: 0X0080
RESET VALUE: 0x00000000
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
RESERVED
DBSTALL
DBIP
DBDOBF
DBDIBF
500
520
530
540
550

FIG. 6

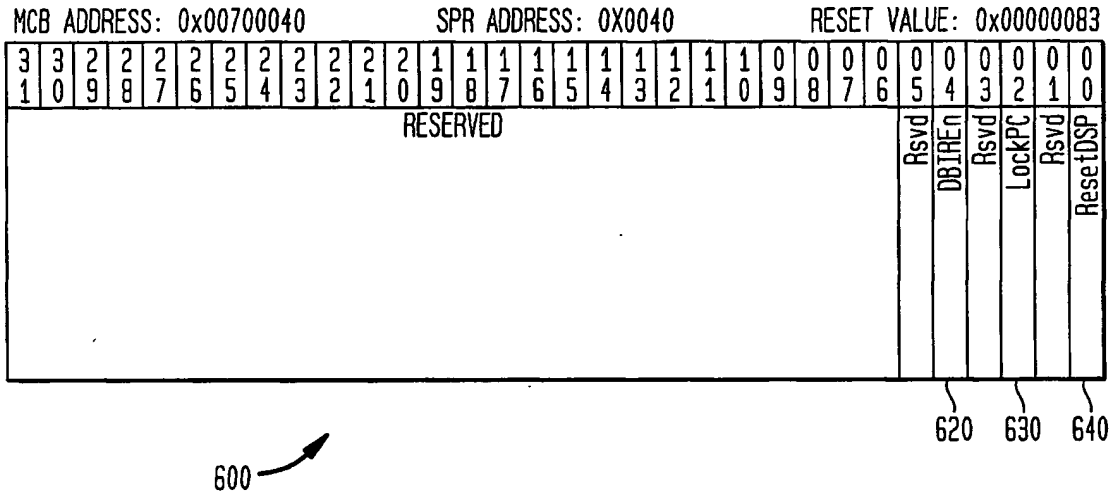

MCB ADDRESS: 0x00700040
SPR ADDRESS: 0X0040
RESET VALUE: 0x00000083
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
RESERVED
Rsvd
DBTREn
Rsvd
LockPC
Rsvd
ResetDSP
620
630
640
600

FIG. 7

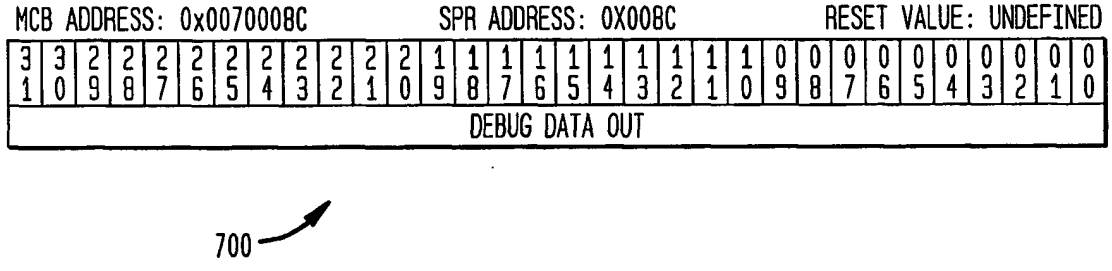

MCB ADDRESS: 0x0070008C
SPR ADDRESS: 0X008C
RESET VALUE: UNDEFINED
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
DEBUG DATA OUT
700

FIG. 8

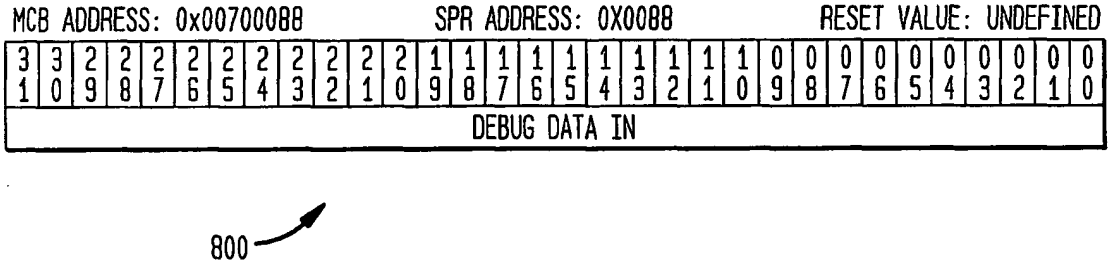

MCB ADDRESS: 0x00700088
SPR ADDRESS: 0X0088
RESET VALUE: UNDEFINED
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
DEBUG DATA IN
800

FIG. 9

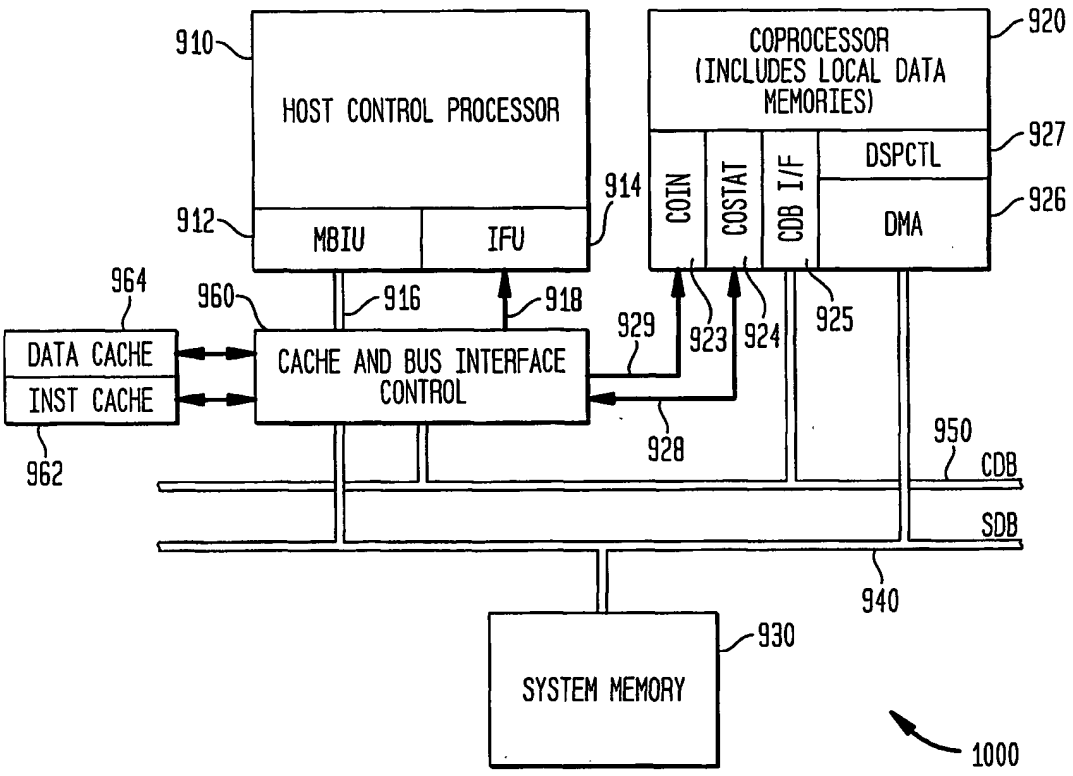

910
HOST CONTROL PROCESSOR
912
MBIU
IFU
914
920
COPROCESSOR (INCLUDES LOCAL DATA MEMORIES)
927
DSPCTL
926
DMA
COIN
COSTAT
CDB I/F
964
960
916
918
929
923
924
925
DATA CACHE
INST CACHE
962
CACHE AND BUS INTERFACE CONTROL
928
950
CDB
SDB
940
930
SYSTEM MEMORY
1000

FIG. 10

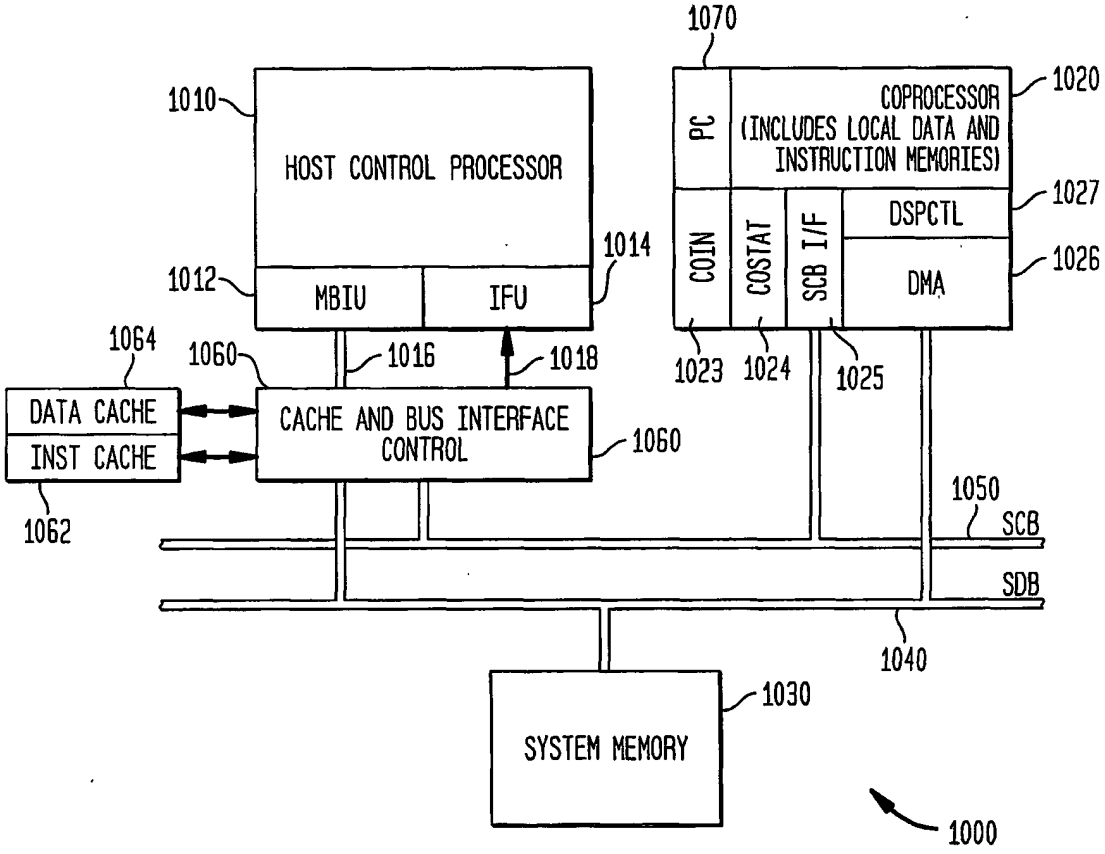
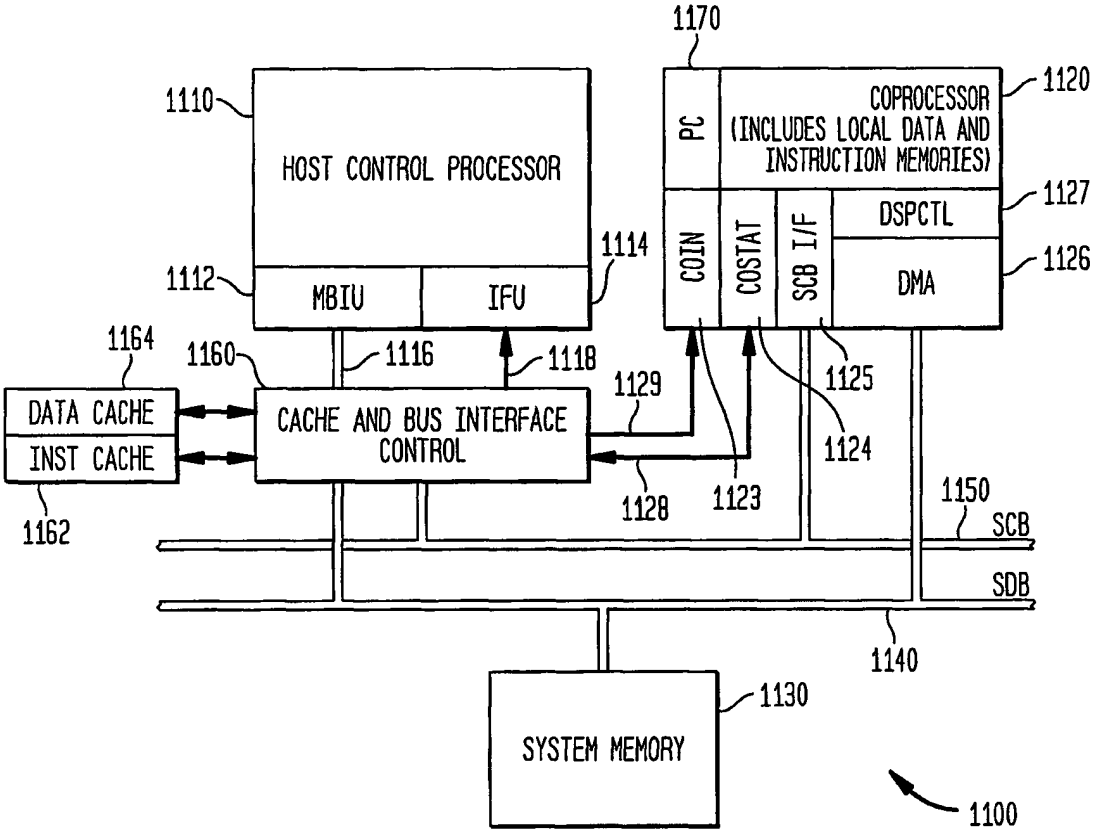

1070
1010
HOST CONTROL PROCESSOR
1012
MBIU
IFU
1014
PC
COPROCESSOR
(INCLUDES LOCAL DATA AND
INSTRUCTION MEMORIES)
1020
COIN
COSTAT
SCB I/F
DSPCTL
1027
DMA
1026
1064
1060
1016
1018
1023
1024
1025
DATA CACHE
INST CACHE
CACHE AND BUS INTERFACE
CONTROL
1060
1062
1050
SCB
SDB
1040
SYSTEM MEMORY
1030
1000

FIG. 11


1170
1110
HOST CONTROL PROCESSOR
PC
COPROCESSOR
(INCLUDES LOCAL DATA AND
INSTRUCTION MEMORIES)
1120
1127
DSPCTL
1126
DMA
COIN
COSTAT
SCB I/F
1114
1112
MBIU
IFU
1164
1160
1116
1118
1129
1125
1124
DATA CACHE
INST CACHE
CACHE AND BUS INTERFACE
CONTROL
1128
1123
1162
1150
SCB
SDB
1140
1130
SYSTEM MEMORY
1100

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/06004

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 15/16
US CL :712/34

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 712/34,35; 714/31, 34

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,682,554 A (HARRELL) 28 OCTOBER 1997.	1-35
X	US 5,890,010 A (NISHIGAMI) 30 MARCH 1999, COL. 3 LINE 15 TO COL. 5 LINE 15.	32-35
X	US 5,996,058 A (SONG ET AL.) 30 NOVEMBER 1999, COL. 1 LINES 36-60.	12-26
X	US 5,442,789 A (BAKER ET AL.) 15 AUGUST 1995, COL. 2 LINES 10-39.	1-11
A	US 5,519,873 A (BUTTER ET AL.) 21 MAY 1996.	1-35
A	US 5,892,897 A (CARLSON ET AL.) 06 APRIL 1999.	1-35
A	US 5,983,018 A (KANZAKI) 09 NOVEMBER 1999.	1-35

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
02 MAY 2001	21 MAY 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer RICHARD ELLIS James R. Matthews Telephone No. (703) 308-3900

Form PCT/ISA/210 (second sheet) (July 1998)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/06004

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5,963,741 A (HORIKAWA) 05 OCTOBER 1999, COL. 4 LINE 30 TO COL. 6 LINE 10.	27-31
A	US 4,943,915 A (WILHELM ET AL.) 24 JULY 1990.	1-35
A	US 4,987,534 A (SEKIGUCHI) 22 JANUARY 1991.	1-35